CS 403X Mobile and Ubiquitous Computing

Lecture 6: Maps, Sensors, Widget Catalog and Presentations

Emmanuel Agu

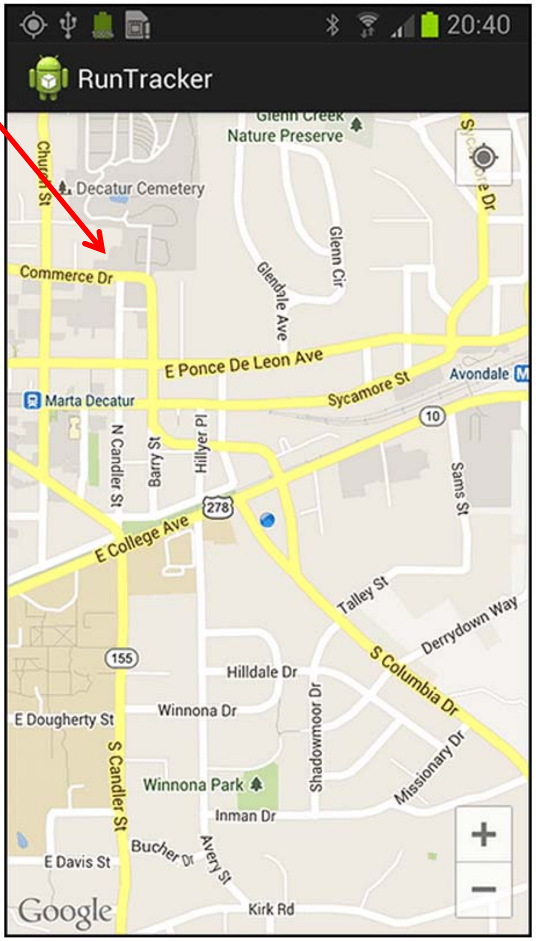
Using Maps

Introducing MapView and Map Activity

- **MapView:** UI widget that displays maps

- **MapActivity:** java class (extends Activity), for displaying maps, handles map-related lifecycle

- **Overlay:** java class used to annotate map, use a canvas to draw unto map layers

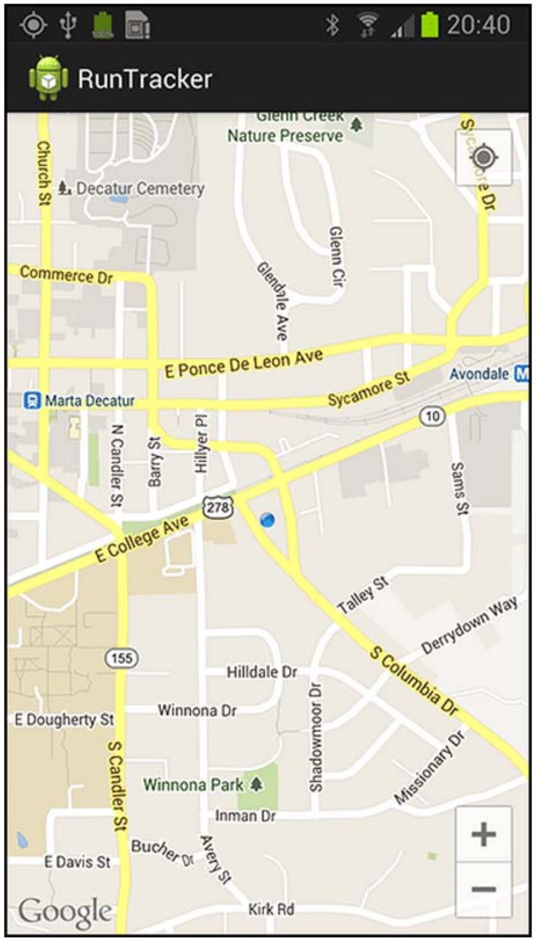


Introducing MapView and Map Activity

- **MapController:** enables map control, setting center location and zoom levels

- **MyLocationOverlay:** Display current location and device orientation

- **ItemizedOverlays and OverlayItems:** used to create markers on map



Steps for using Google Maps Android API v2

1. Install Android SDK (Done already!)
2. Download and configure **Google Play services** SDK, which includes Google Maps API
3. Obtain an API key
4. Add required settings (permissions, etc) to Android Manifest
5. Add a map to app
6. Publish your app (optional)

Ref: https://developers.google.com/maps/documentation/android/start#getting_the_google_maps_android_api_v2

Step 2: Install & Configure Google Play Services SDK

- Google Maps API v2 is part of Google Services SDK

- Main steps to set up Google Play Services
  - Install Google Play services SDK
  - Add Google Play services as an Android library project
  - Reference the Google Play services in your app's project
  - See: https://developer.android.com/google/play-services/setup.html

Step 3: Get Google Maps API key

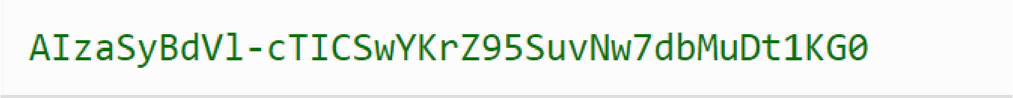
- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free
- **Background:** Before they can be installed, android apps must be **signed with digital certificate** (developer holds private key)
- Digital certificates uniquely identify an app, used in tracking:
    - Apps within Google Play Store and
    - App's use of resources such as Google Map servers
- Android apps often use **self-signed certificates**, not authority
- **See: http://developer.android.com/tools/publishing/app-signing.html**

Step 3: Get Google Maps API key (Contd)

- To obtain a Maps API key, app developer provides:
  - App’s signing certificate + its package name
- **Certificate/package pairs => Maps API keys**
- Steps to obtain a Maps API requires following steps
  - Retrieve information about app’s certificate
  - Register a project in Google APIs console and add the Maps API as a service for the project
  - Request one or more keys
  - Add key to app and begin development
  - **See: https://developers.google.com/maps/documentation/android/start**

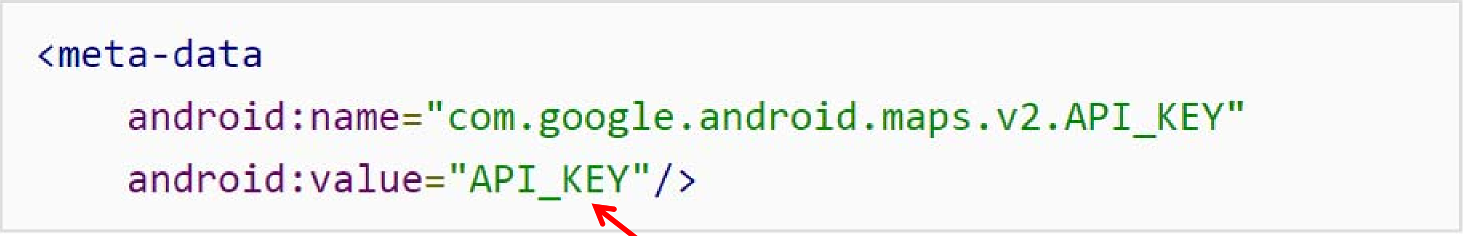
Step 3: Get Google Maps API key (Contd)

- If successful, 40-character API key generated, for example



```
AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0
```

- Add this API key to app in order to use Maps API
- Include API key in AndroidManifest.xml
- To modify AndroidManifest.xml, add following between <application> … </application>



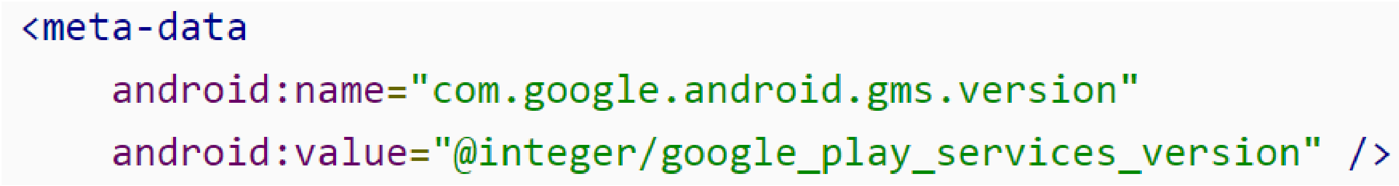
```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY"/>
```

Insert Maps API key here
Makes API key visible to any MapFragment in app

- Maps API reads key value from AndroidManifest.xml, passes it to Google Maps server to authenticate access

Step 4: Add Settings to AndroidManifest.xml

- Add Google Play services version to AndroidManifest.xml



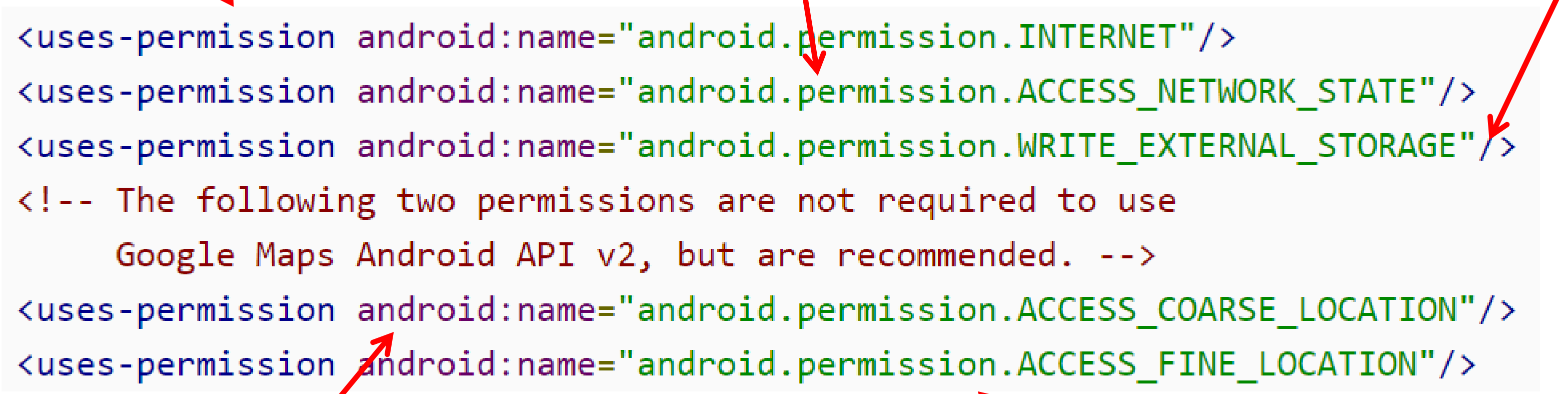
```xml
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

- Request the following permissions:

Used by API to download map tiles from Google Maps servers

Allows the API to check the connection status to determine if data can downloaded

Used by API to cache map tile data in device’s external storage



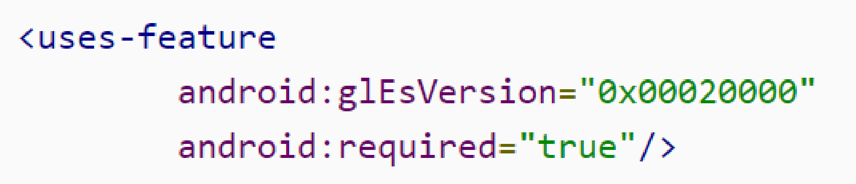
```xml
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- The following two permissions are not required to use
     Google Maps Android API v2, but are recommended. -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Allows API to use WiFI or mobile cell data (or both) to determine the device’s location

Allows the API to use GPS to determine device’s location within a small area

Step 4: Add Settings to AndroidManifest.xml (Contd)

- Specify that OpenGL ES version 2 is required
- Why? Google Maps Android API uses OpenGL ES version 2 to render the map

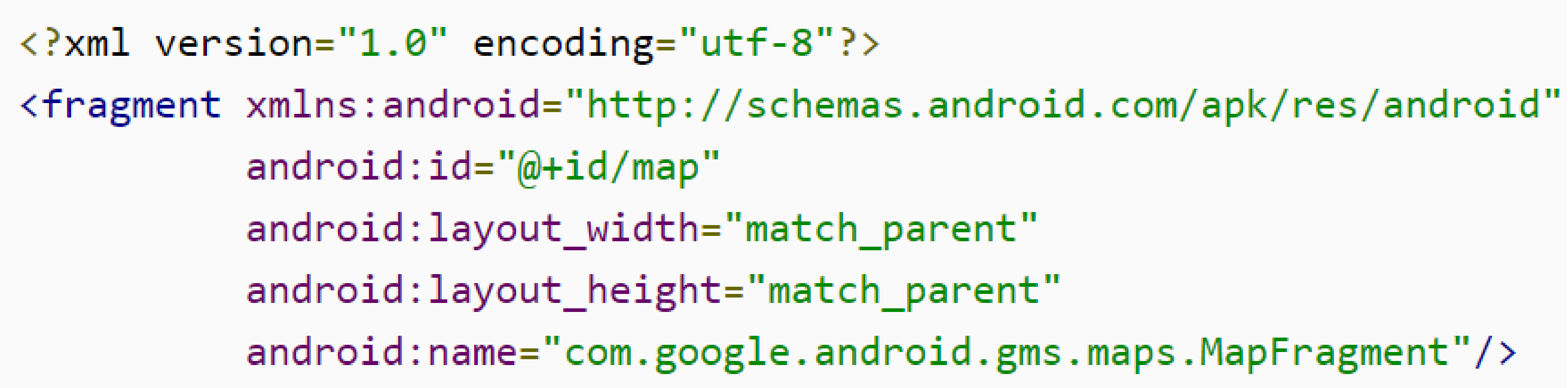


```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

- Due to above declaration, devices that don’t have OpenGL ES version 2 will not see the app on Google Play

Step 5: Add a map

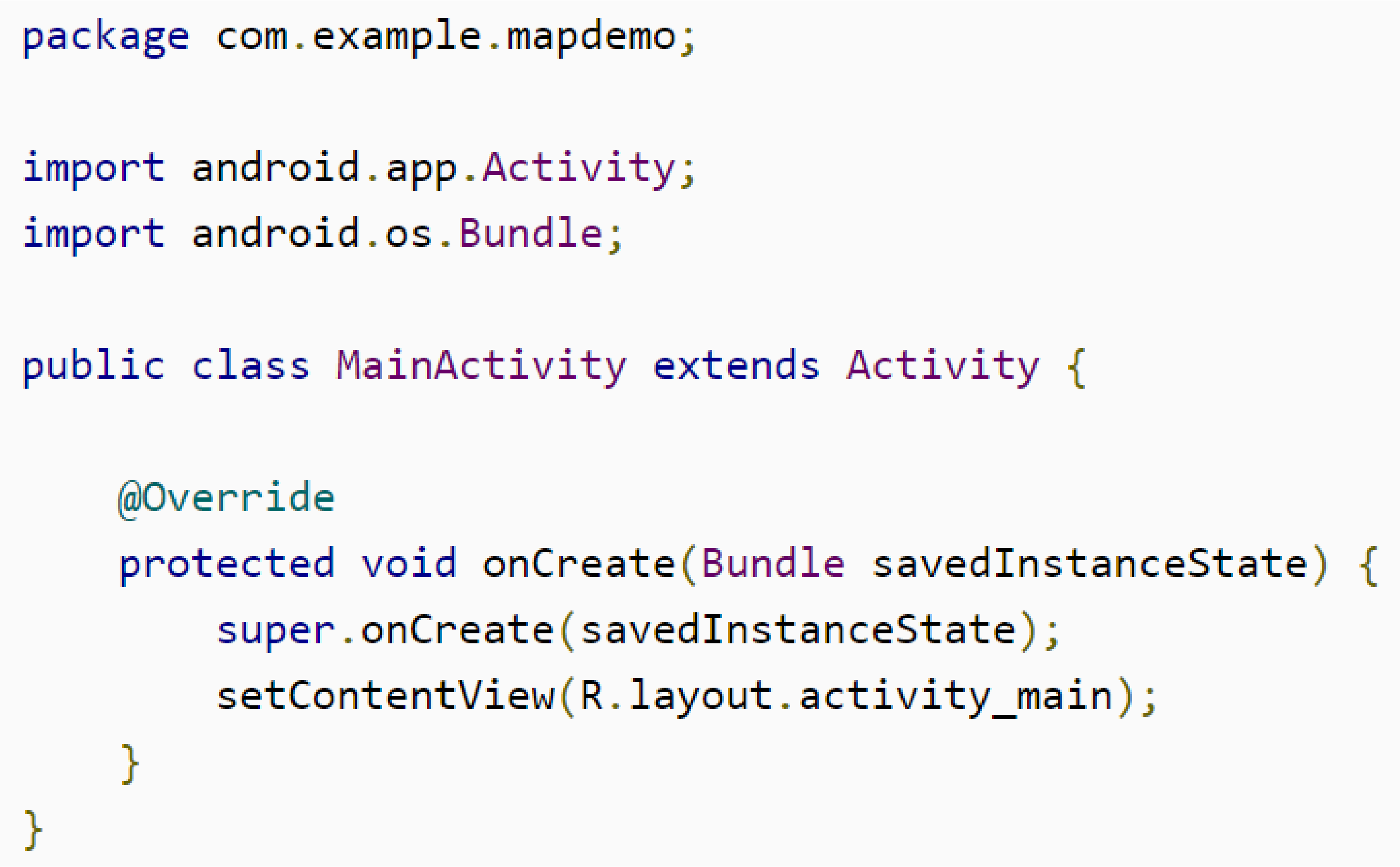
- Map is generated as fragment within an activity
- To add map, add following to XML layout file



```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
```

Install & Configure Google Play Services SDK

- And create MainActivity.java as usual

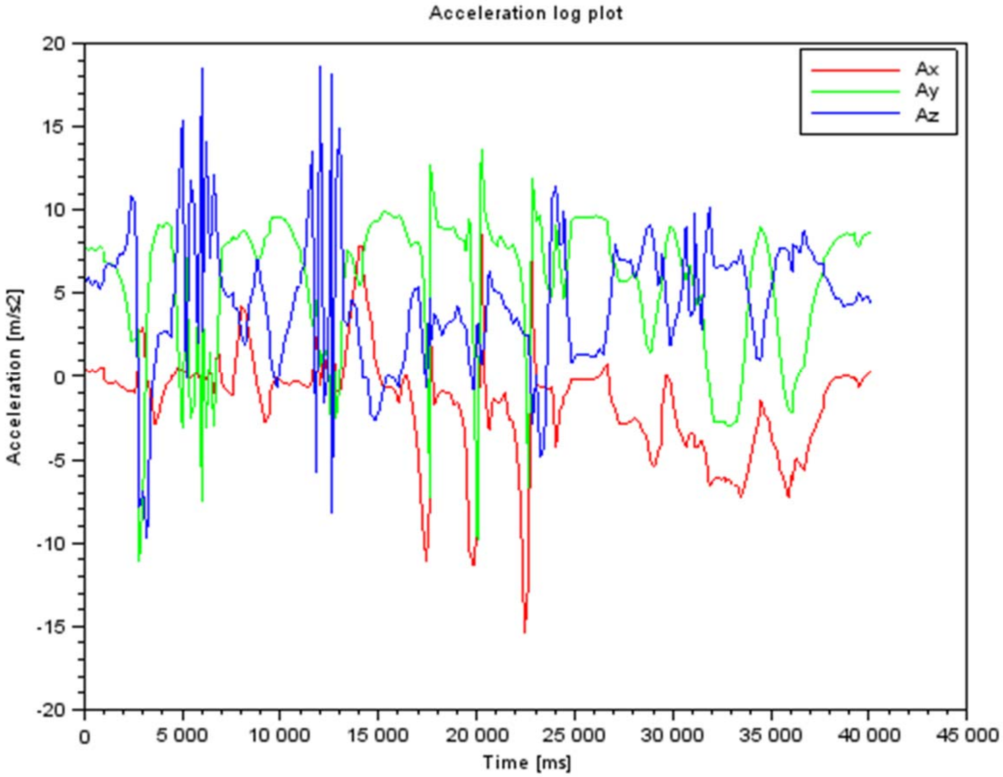


```java
package com.example.mapdemo;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Sensors

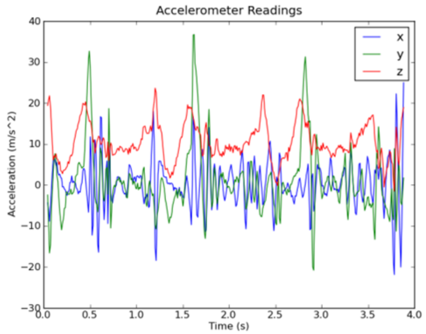
What is a Sensor?

- Converts some physical quantity (e.g. light, acceleration, magnetic field) into a signal
- **Example:** accelerometer converts acceleration along X,Y,Z axes into signal



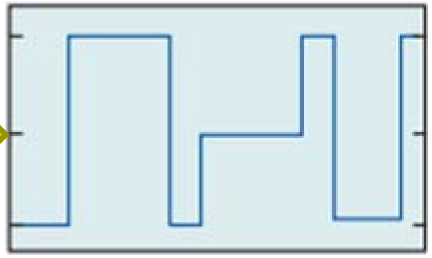
So What?

- Raw sensor data can be processed into meaningful info
- Example: Raw accelerometer data can be processed to infer user's activity (e.g. walking running, etc)



Raw accelerometer Readings

+ Activity labels



Machine learning Feature extraction and classification

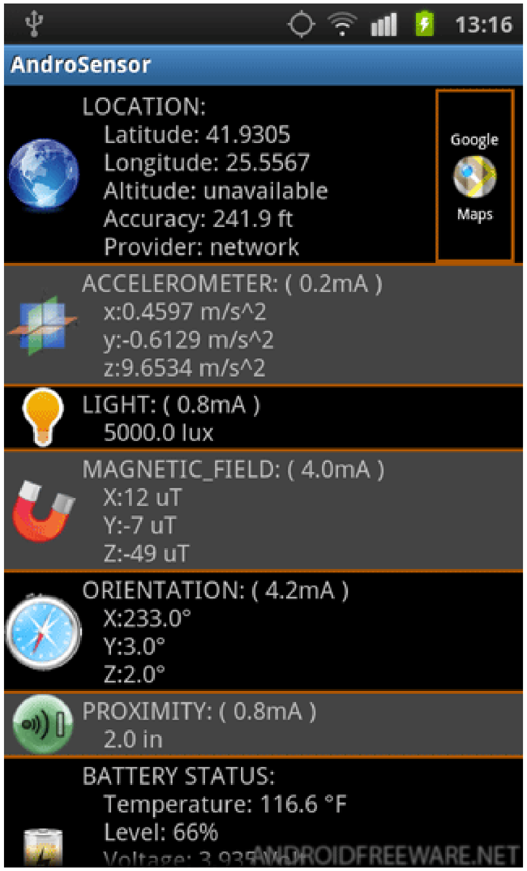
(e.g using Weka)

Walking
Running
Jumping
Step count
Calories burned
Falling

Android Sensors

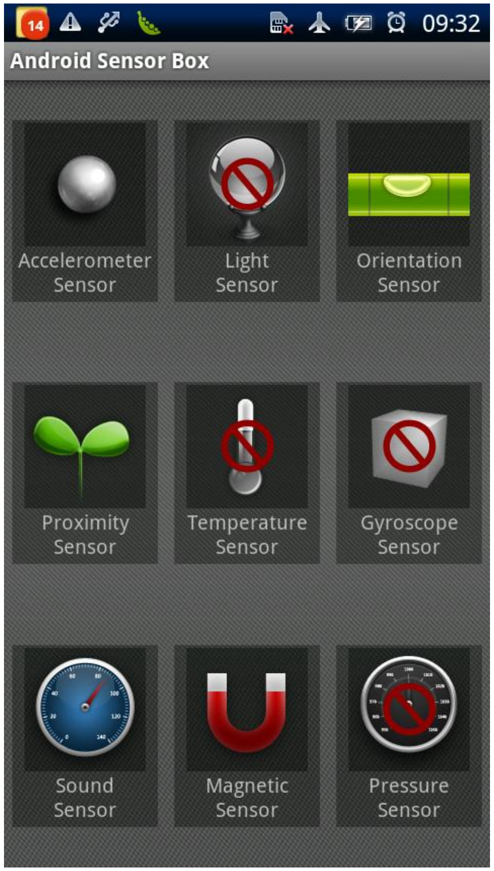
- Microphone (sound)
- Camera
- Temperature
- Location
- Accelerometer
- Gyroscopy (orientation)
- Proximity
- Pressure
- Light

- **Different phones have different set of sensors!!!**



```
13:16
AndroSensor
LOCATION:
  Latitude: 41.9305
  Longitude: 25.5567
  Altitude: unavailable
  Accuracy: 241.9 ft
  Provider: network
ACCELEROMETER: ( 0.2mA )
  x:0.4597 m/s^2
  y:-0.6129 m/s^2
  z:9.6534 m/s^2
LIGHT: ( 0.8mA )
  5000.0 lux
MAGNETIC_FIELD: ( 4.0mA )
  X:12 uT
  Y:-7 uT
  Z:-49 uT
ORIENTATION: ( 4.2mA )
  X:233.0°
  Y:3.0°
  Z:2.0°
PROXIMITY: ( 0.8mA )
  2.0 in
BATTERY STATUS:
  Temperature: 116.6 °F
  Level: 66%
  Voltage: 3.93
```

AndroSensor



Android Sensor Box

Android Sensor Framework

- Enables apps to:
  - Access sensors available on device and
  - Acquire raw sensor data

- Specifically, using the Android Sensor Framework, you can:
  - Determine which sensors are available
  - Determine capabilities of individual sensors (e.g. max. range, manufacturer, power requirements, resolution)
  - Acquire raw sensor data and define data rate
  - Register and unregister sensor event listeners

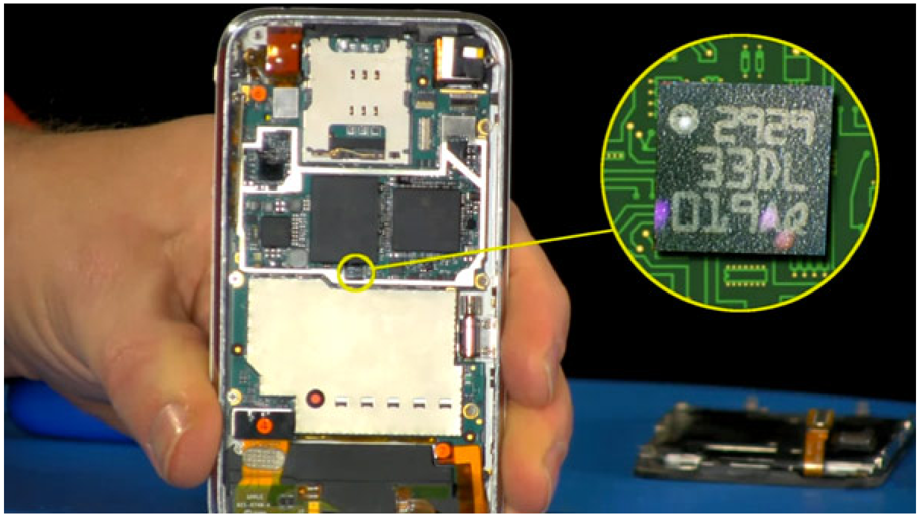
http://developer.android.com/guide/topics/sensors/sensors_overview.html

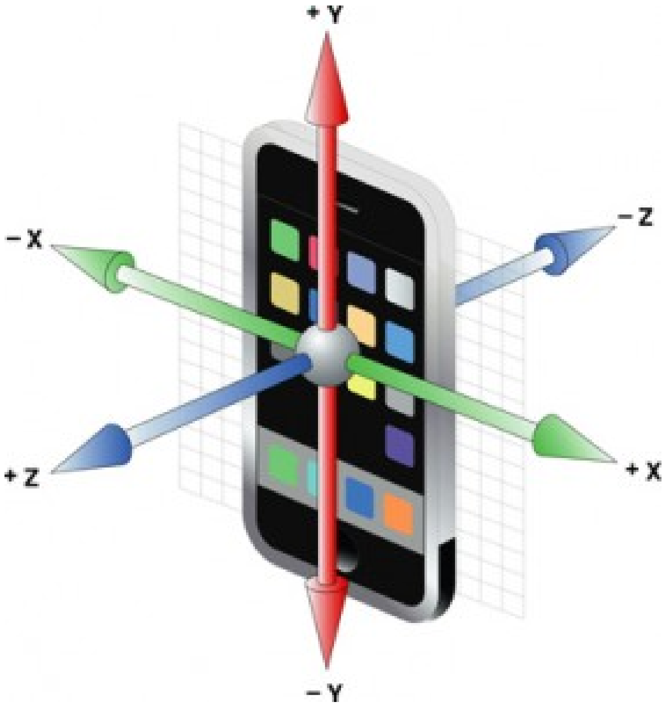
Android Sensor Framework

- Android sensors can be either hardware or software
- **Hardware sensor:**
  - physical components built into phone,
  - Measure specific environmental property. **E.g. temperature**
- **Software sensor (or virtual sensor):**
  - Not physical device
  - Derives their data from one or more hardware sensors
  - **Example:** gravity sensor

Accelerometer Sensor

- Acceleration is **rate of change of velocity**
- Accelerometers
  - Measure **change** of speed in a direction
  - Do not measure velocity
- Phone’s accelerometer measures acceleration along its X,Y,Z axes





Sensor Types Supported by Android

- TYPE_ACCELEROMETER
  - **Type:** hardware
  - Measures device acceleration force along X,Y,Z axes **including gravity** in m/s^2
  - **Common uses:** motion detection (shake, tilt, etc)

- TYPE_LINEAR_ACCELEROMETER
  - **Type:** software or hardware
  - Measures device acceleration force along X,Y,Z axes **excluding gravity** in m/s^2
  - **Common uses:** monitoring acceleration along single axis

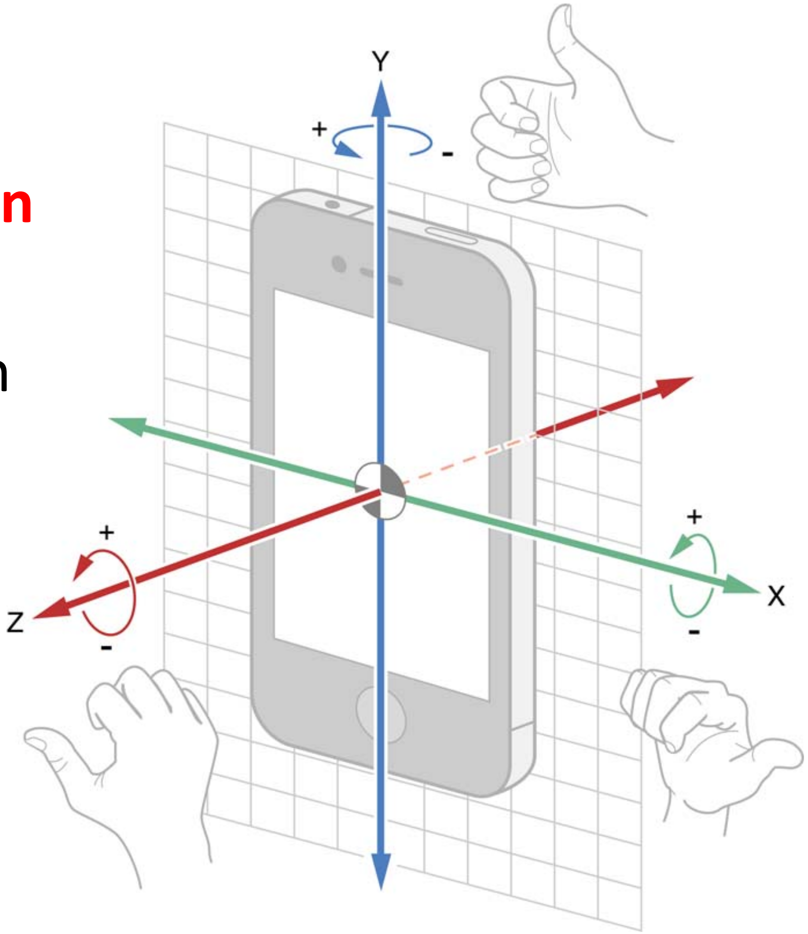
Sensor Types Supported by Android

- TYPE_GRAVITY
  - **Type:** Software or hardware
  - Measures the force of **gravity** along X,Y,Z axes in m/s^2
  - **Common uses:** motion detection (shake, tilt, etc)

- TYPE_ROTATION_VECTOR
  - **Type:** Software or hardware
  - Measures **device's orientation** by providing 3 rotation vectors
  - **Common uses:** motion detection and rotation

Sensor Types Supported by Android

- TYPE_GYROSCOPE
  - **Type:** hardware
  - Measures device's **rate of rotation** around X,Y,Z axes in rad/s
  - **Common uses:** rotation detection (spin, turn, etc)



Sensor Types Supported by Android

- TYPE_AMBIENT_TEMPERATURE
  - **Type:** hardware
  - Measures ambient **room temperature** in degrees Celcius
  - **Common uses:** monitoring room air temperatures

- TYPE_LIGHT
  - **Type:** hardware
  - Measures ambient **light level** (illumination) in lux
  - Lux is SI measure of illuminance
    - Measures luminous flux per unit area
  - **Common uses:** controlling screen brightness

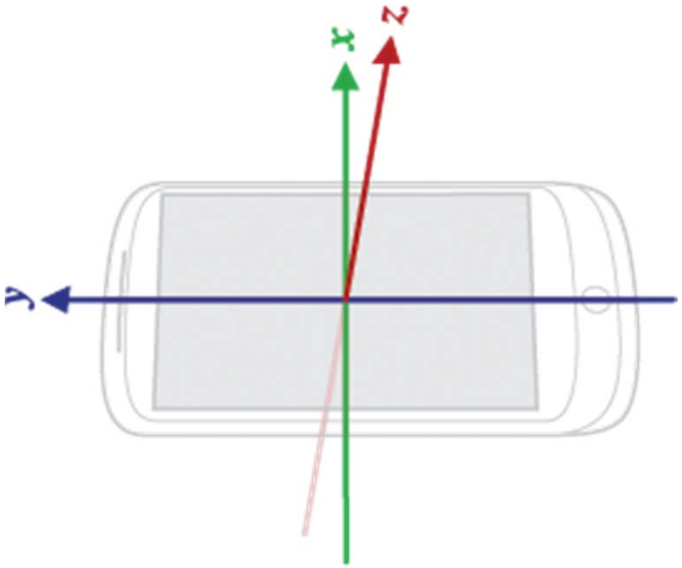
Sensor Types Supported by Android

- TYPE_MAGNETIC_FIELD
  - **Type:** hardware
  - Measures **magnetic field** for X,Y,Z axes in μT
  - **Common uses:** Creating a compass

- TYPE_PRESSURE
  - **Type:** hardware
  - Measures ambient **air pressure** in hPa or mbar
  - Force per unit area
  - **Common uses:** monitoring air pressure changes

Sensor Types Supported by Android

- TYPE_ORIENTATION
  - **Type:** software
  - Measures degrees of rotation about X,Y,Z axes
  - **Common uses:** Determining device position



Sensor Types Supported by Android

- TYPE_PROXIMITY
  - **Type:** hardware
  - Measures proximity of an object in cm relative to view device's screen.
  - **Common uses:** to determine whether a handset is being held up to a person's ear

Sensor Types Supported by Android

- TYPE_RELATIVE HUMIDITY
  - **Type:** hardware
  - Measures relative ambient humidity in percent (%)
  - **Relative humidity:** % of max possible humidity present in air
  - **Common uses:** monitoring dewpoint, absolute, and relative humidity

2 New Hardware Sensor in Android 4.4

- TYPE_STEP_DETECTOR
  - **Type:** hardware
  - Triggers a sensor event each time user takes a step
  - Delivered event has value of 1.0 and timestamp of step

- TYPE_STEP_COUNTER
  - **Type:** hardware
  - Also triggers a sensor event each time user takes a step
  - Delivers total ***accumulated number of steps since this sensor was first registered by an app,*** tries to eliminate false positives
- **Common uses:** Both used in step counting, pedometer apps
- Requires hardware support, available in Nexus 5

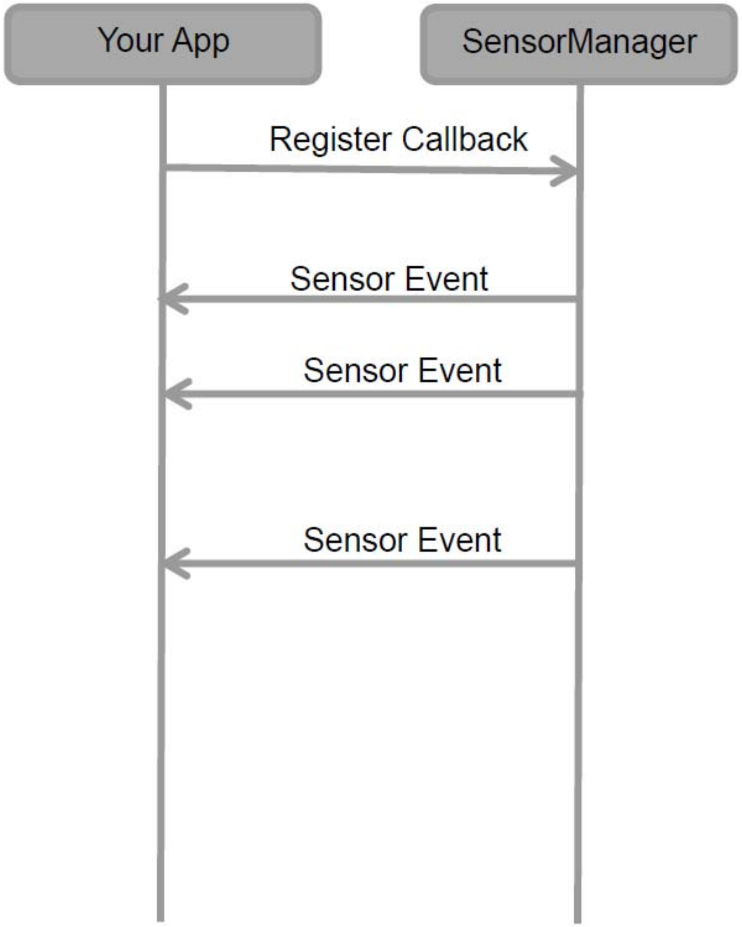
Sensor Programming

- Sensor framework is part of **android.hardware**
- Classes and interfaces include:
  - **SensorEvent** ←
  - **Sensor**
  - **SensorEventListener**
  - **SensorManager**
- These sensor-APIs used for 2 main tasks:
  - Identifying sensors and sensor capabilities
  - Monitoring sensor events

Sensor Events and Callbacks

- App sensors send events asynchronously, when new data arrives

- General approach:
  - App registers callbacks
  - SensorManager notifies app of sensor event whenever new data arrives (or accuracy changes)



Recall: Sensor Programming

- Sensor classes and interfaces include:
  - **SensorEvent**
  - **Sensor** ←
  - **SensorEventListener**
  - **SensorManager**

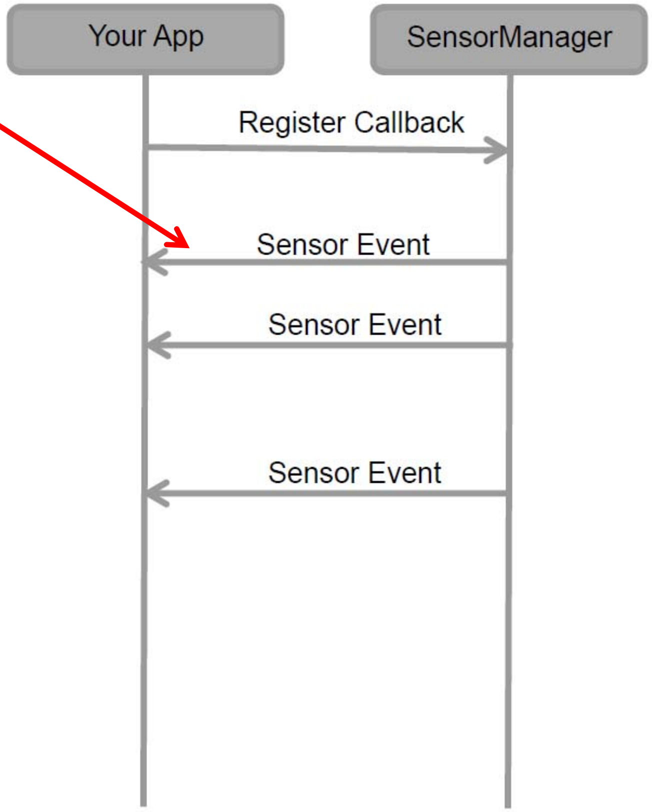
Sensor

- A class that provides methods used to determine a sensor’s capabilities
- Can be used to create instance of a specific sensor
- E.g. create instance of accelerometer

SensorEvent Object

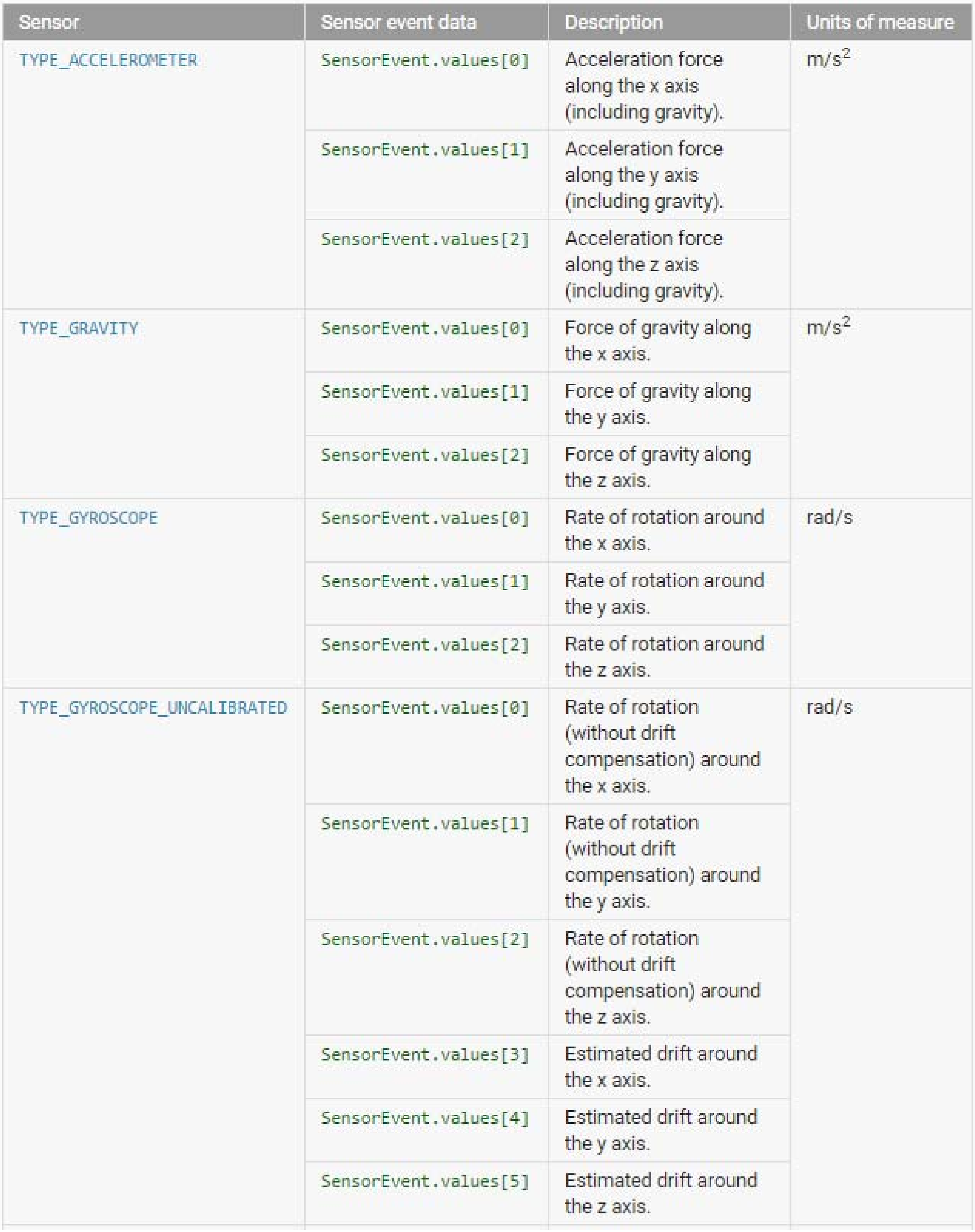
- Android system provides information about a sensor event as a **sensor event object**

- **Sensor event object** includes:
  - ***Values:*** Raw sensor data
  - ***Sensor:*** Type of sensor that generated the event
  - ***Accuracy:*** Accuracy of the data
  - ***Timestamp:*** Event timestamp



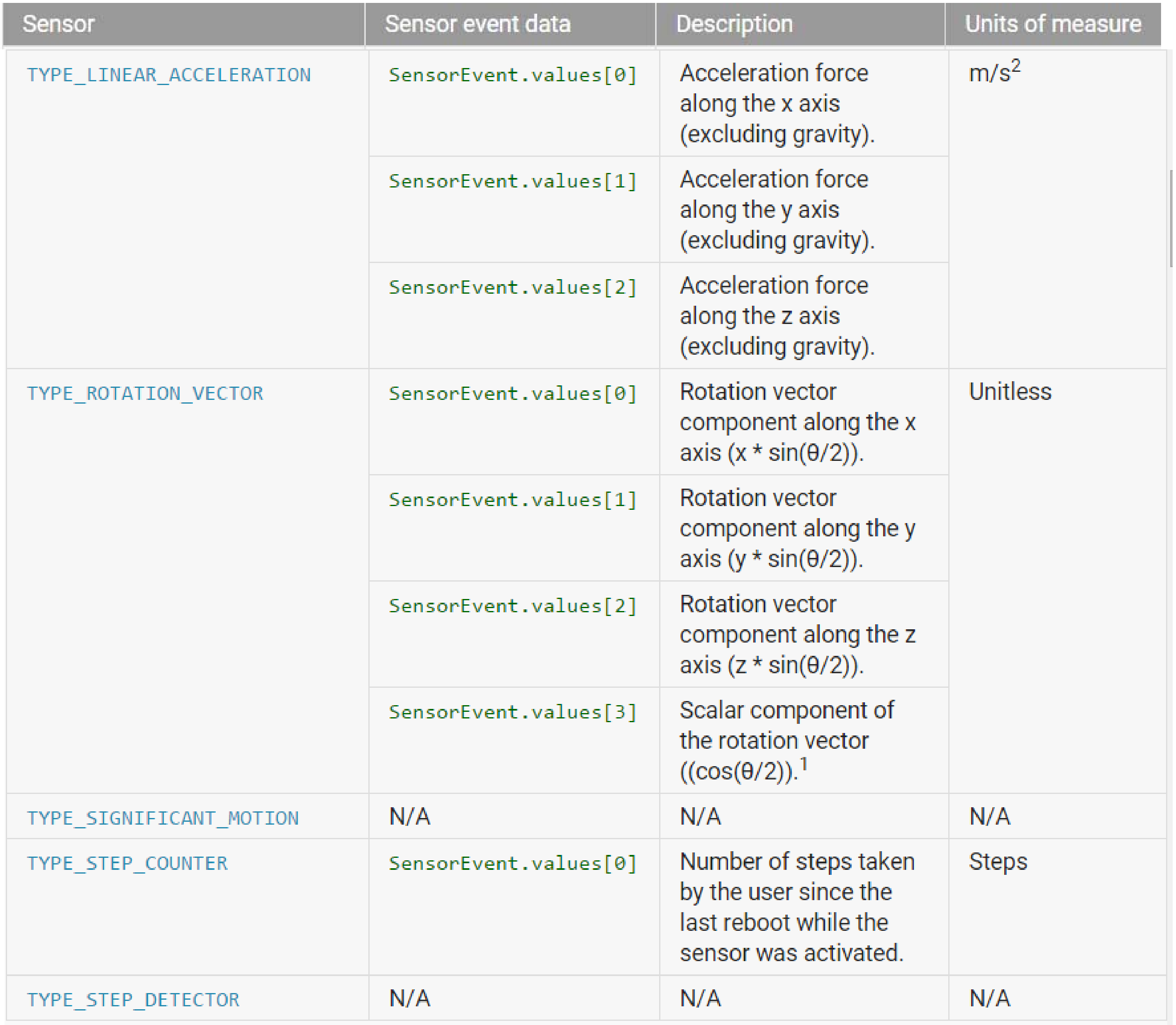
Sensor values?

Sensor Values Depend on Sensor Type



Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s^2
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	
TYPE_GYROSCOPE_UNCALIBRATED	SensorEvent.values[0]	Rate of rotation (without drift compensation) around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation (without drift compensation) around the y axis.	
	SensorEvent.values[2]	Rate of rotation (without drift compensation) around the z axis.	
	SensorEvent.values[3]	Estimated drift around the x axis.	
	SensorEvent.values[4]	Estimated drift around the y axis.	
	SensorEvent.values[5]	Estimated drift around the z axis.	

Sensor Values Depend on Sensor Type



Sensor	Sensor event data	Description	Units of measure
TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).[1]	
TYPE_SIGNIFICANT_MOTION	N/A	N/A	N/A
TYPE_STEP_COUNTER	SensorEvent.values[0]	Number of steps taken by the user since the last reboot while the sensor was activated.	Steps
TYPE_STEP_DETECTOR	N/A	N/A	N/A

Recall: Sensor Programming

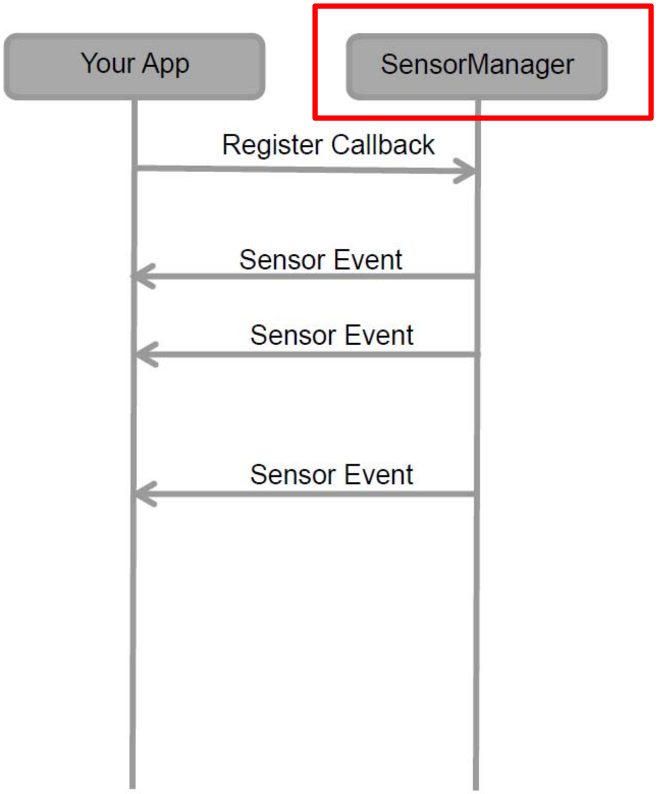
- Sensor classes and interfaces include:
  - **SensorEvent**
  - **Sensor**
  - **SensorEventListener** ←
  - **SensorManager**

SensorEventListener

- An interface used to create 2 callbacks that receive notifications (sensor events) when:
  - Sensor **values change** **(onSensorChange( ))** or
  - When sensor **accuracy changes** **(onAccuracyChanged( ))**

Recall: Sensor Programming

- Sensor classes and interfaces include:
  - **SensorEvent**
  - **Sensor**
  - **SensorEventListener**
  - **SensorManager** ←

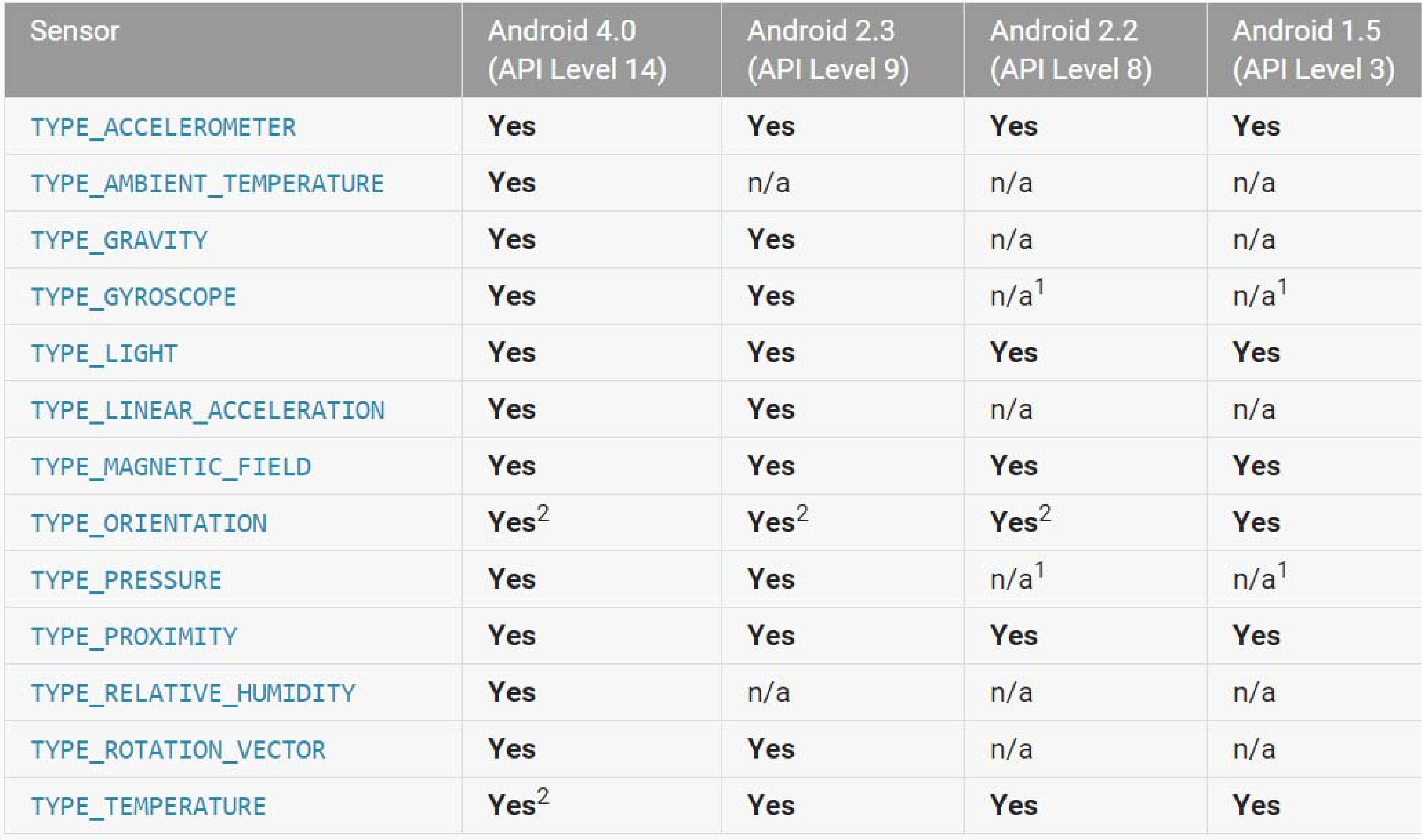


SensorManager

- A class that provides methods for:
  - Accessing and listing sensors
  - Registering and unregistering sensor event listeners
  - Acquiring orientation information
- Can be used to create instance of **sensor service**
- Also provides sensor **constants** used to:
  - Report sensor ***accuracy***
  - Set ***data acquisition rates***
  - Calibrate sensors

Sensor Availability

- Different sensors are available on different Android versions



Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a[1]	n/a[1]
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes[2]	Yes[2]	Yes[2]	Yes
TYPE_PRESSURE	Yes	Yes	n/a[1]	n/a[1]
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes[2]	Yes	Yes	Yes

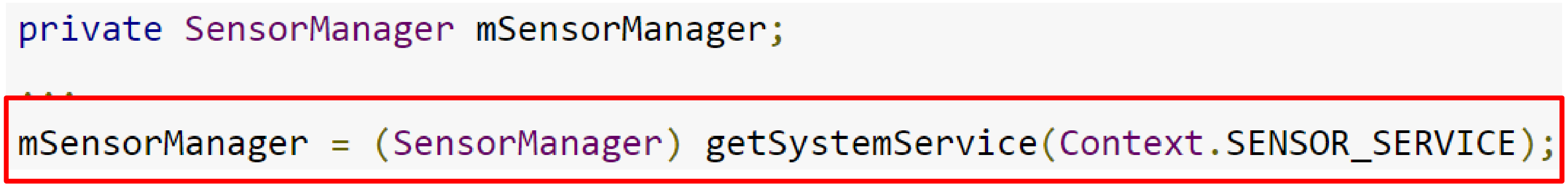
Sensor API Tasks

- **Sensor API Task 1: Identifying sensors and their capabilities**
- Why identify sensor and their capabilities at runtime?
  - Disable app features using sensors not present, or
  - Choose sensor implementation with best performance

- **Sensor API Task 2: Monitor sensor events**
- Why monitor sensor events?
  - To acquire raw sensor data
  - Sensor event occurs every time sensor detects change in parameters it is measuring (in physical world. E.g. temperature changes)

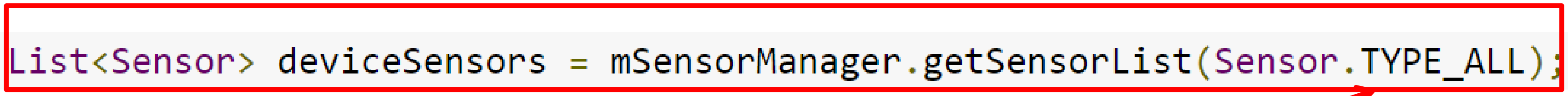
Identifying Sensors and Sensor Capabilities

- Need a reference to the sensor service.
- How? First create instance of **SensorManager** by calling **getSystemService( )** and passing in SENSOR_SERVICE argument



```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- Then list sensors available on device by calling **getSensorList( )**



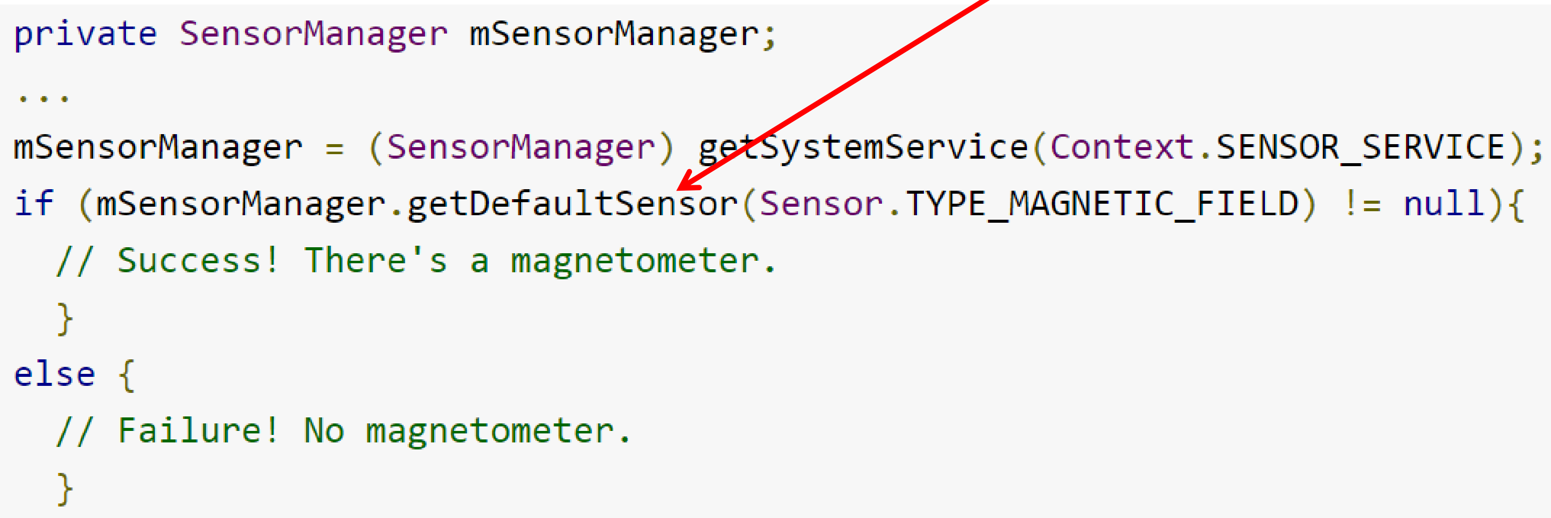
```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- To list particular type, use **TYPE_GYROSCOPE, TYPE_GRAVITY**, etc

http://developer.android.com/guide/topics/sensors/sensors_overview.html

Determing if Device has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
  - E.g. multiple magnetometers
- If multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use **getDefaultSensor( )**
- **Example:** To check whether device has a magnetometer



```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
  // Success! There's a magnetometer.
  }
else {
  // Failure! No magnetometer.
  }
```

Determining Capabilities of Sensors

- Some useful methods of **Sensor** class methods:
  - **getResolution( ):** get sensor’s resolution
  - **getMaximumRange( ):** get maximum measurement range
  - **getPower( ):** get sensor’s power requirements
  - **getMinDelay( ):** min time interval (in microseconds) sensor can use to sense data. Return values:
    - **0 value:** Non-streaming sensor, reports data only if sensed parameters change
    - **Non-zero value:** streaming sensor

Monitoring Sensor Events

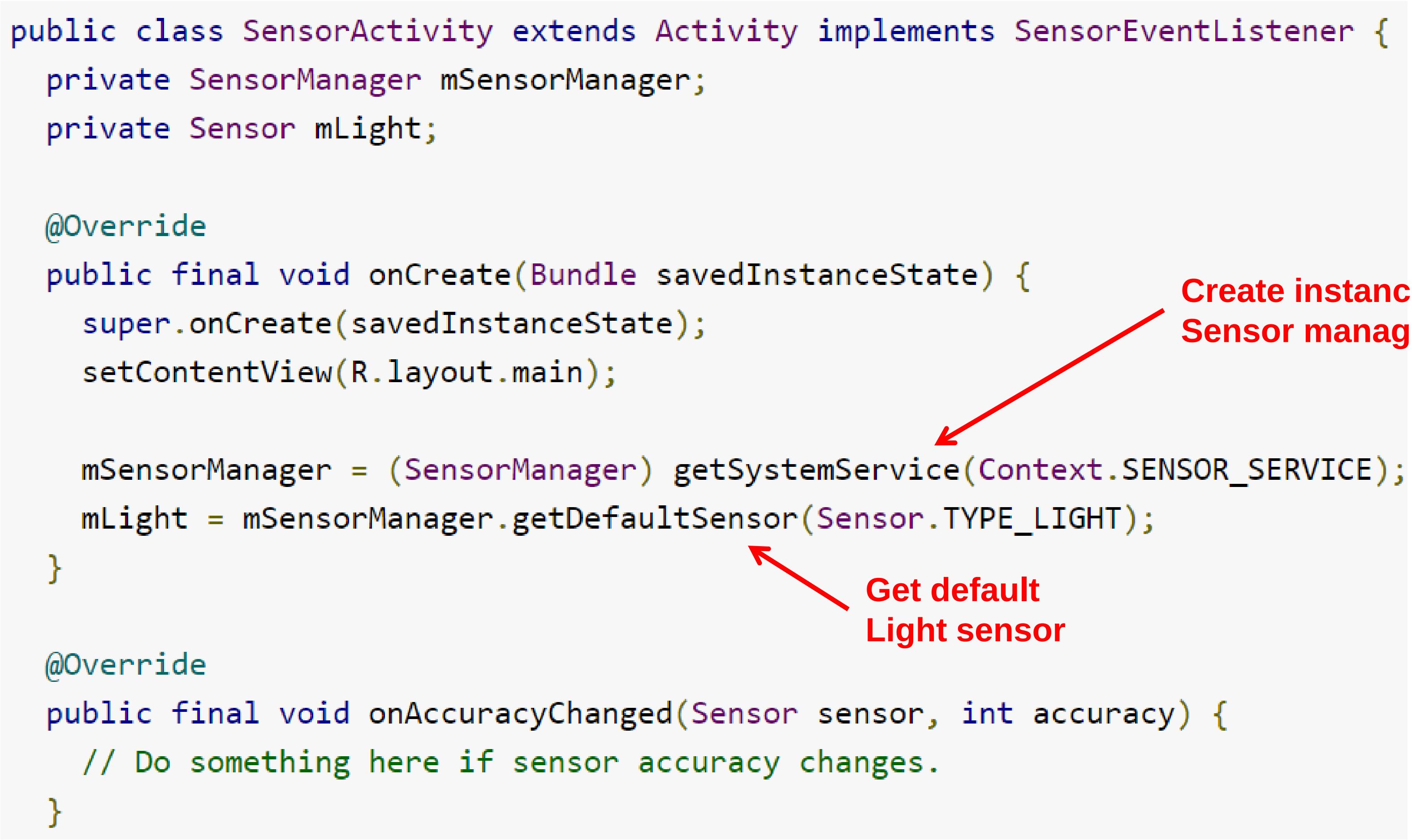
- To monitor raw sensor data, 2 callback methods exposed through **SensorEventListener** interface need to be implemented:
- **onSensorChanged:**
  - Called by Android OS to report **new sensor value**
  - Passes **SensorEvent** object containing information about new sensor data to your app
  - New sensor data includes:
    - **Accuracy:** Accuracy of data
    - **Sensor:** Sensor that generated the data
    - **Timestamp:** Times when data was generated
    - **Data:** New data that sensor recorded

Monitoring Sensor Events

- **onAccuracyChanged:**
  - invoked when **accuracy of sensor** being monitored changes
  - Provides reference to **sensor object** that changed and the new accuracy of the sensor
  - Accuracy represented as status constants SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM,
  - SENSOR_STATUS_ACCURACY_HIGH,
  - SENSOR_STATUS_UNRELIABLE

Example: Monitoring Light Sensor Data

- **Goal:** Monitor light sensor data using **onSensorChanged( )**, display it in a **TextView** defined in main.xml

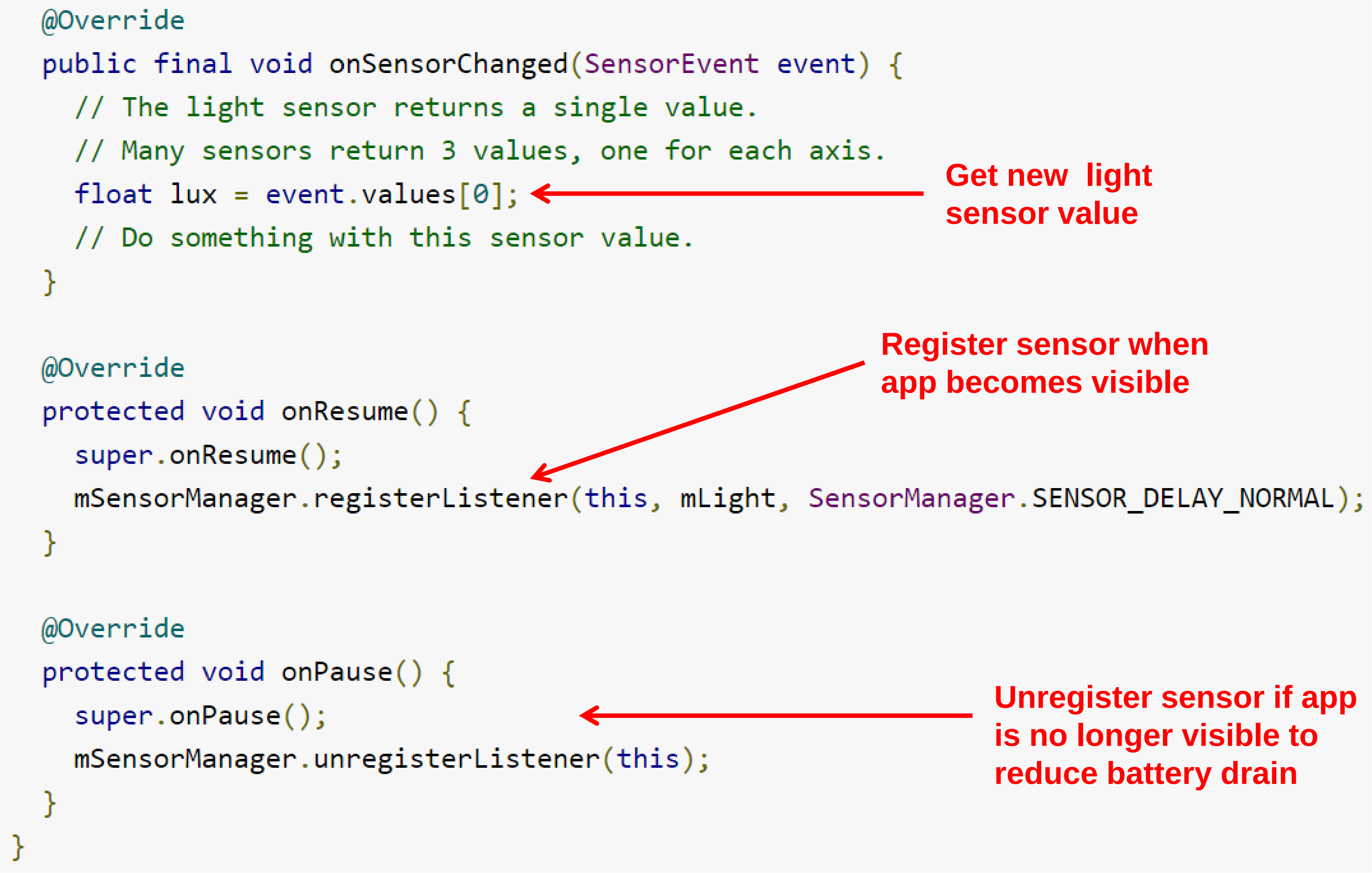


```
public class SensorActivity extends Activity implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
  }
```

Create instance of Sensor manager

Get default Light sensor

Example: Monitoring Light Sensor Data (Contd)



```
  @Override
  public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
  }

  @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
}
```

Get new light sensor value

Register sensor when app becomes visible

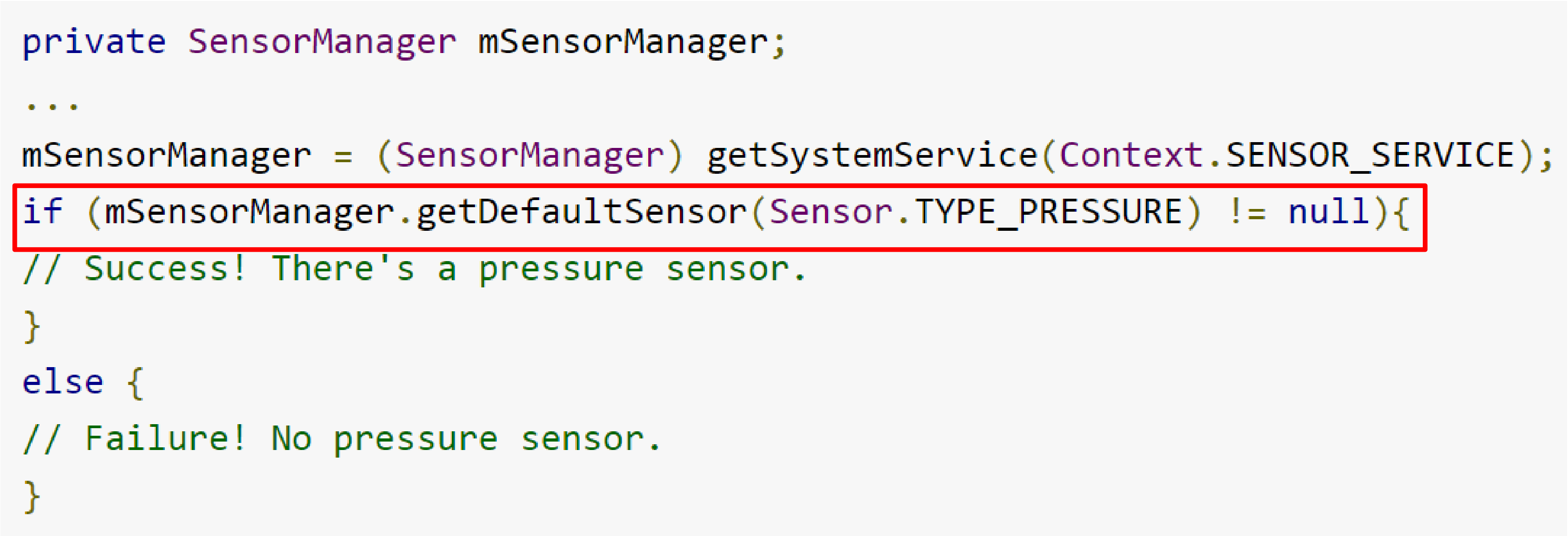
Unregister sensor if app is no longer visible to reduce battery drain

Handling Different Sensor Configurations

- Different phones have different sensors built in
- **E.g.** Motorola Xoom has pressure sensor, Samsung Nexus S doesn't
- If app uses a specific sensor, how to ensure this sensor exists on target device? Two options
  - **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
  - **Option 2:** Use Google Play filters so only devices possessing required sensor can download app

Option 1: Detecting Sensors at Runtime

- Following code checks if device has a pressure sensor

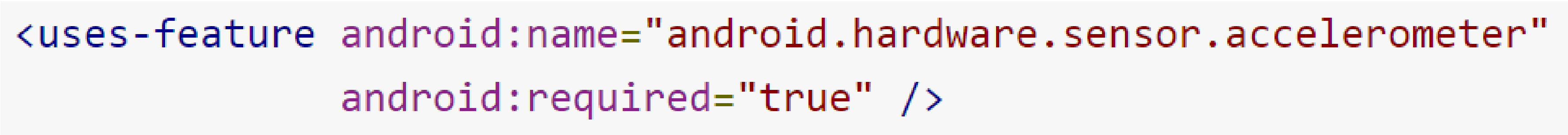


```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
// Success! There's a pressure sensor.
}
else {
// Failure! No pressure sensor.
}
```

Option 2: Use Google Play Filters to Target Specific Sensor Configurations

- Can use **<uses-feature>** element in AndroidManifest.xml to filter your app from devices without required sensors

- **Example:** following manifest entry ensures that only devices with accelerometers will see this app on Google Play

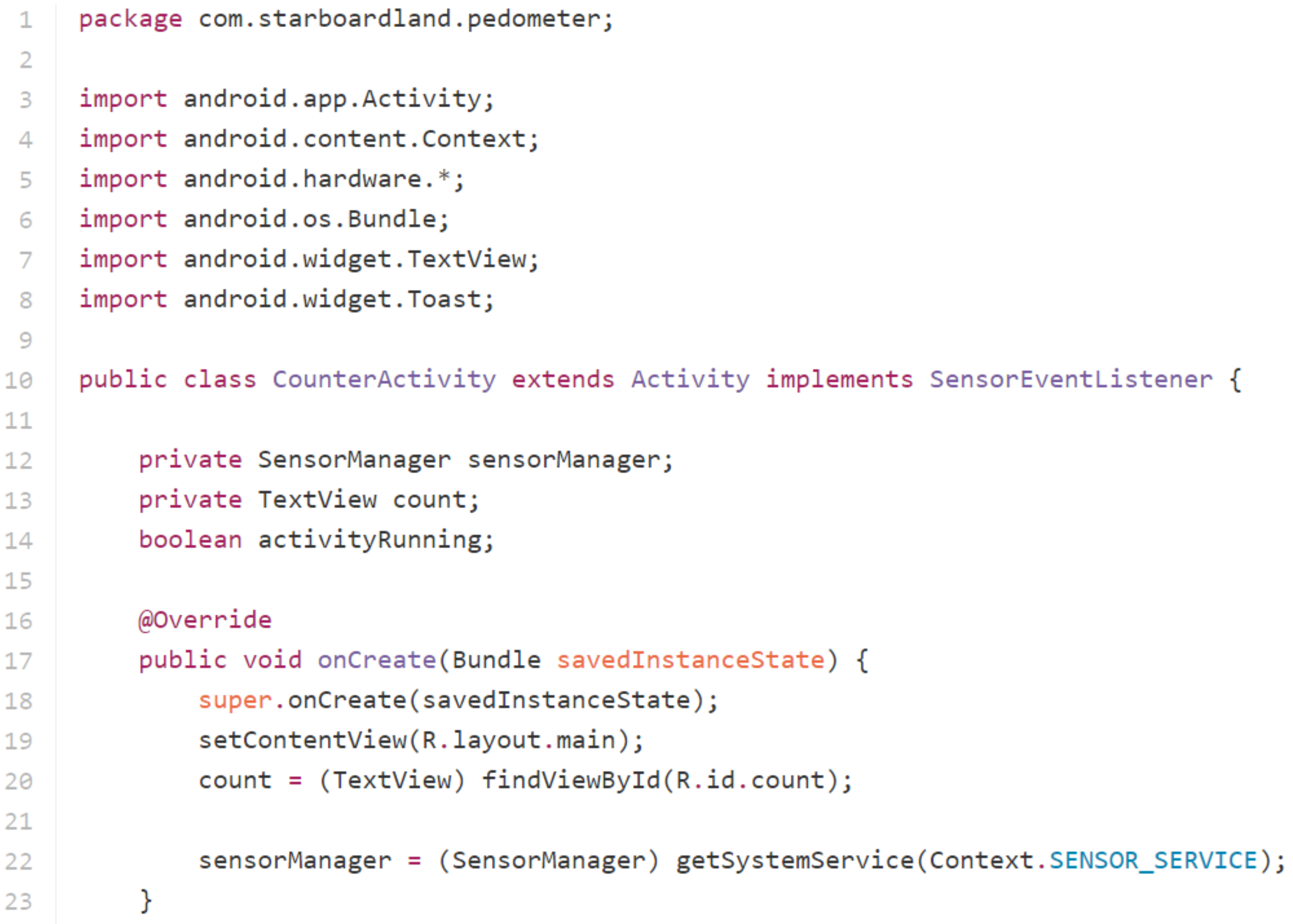


```
<uses-feature android:name="android.hardware.sensor.accelerometer"
              android:required="true" />
```

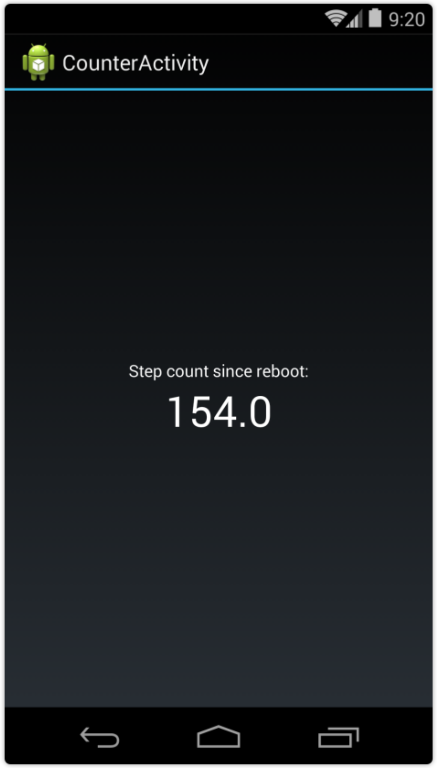
- **Can list** accelerometers, barometers, compass (geomagnetic field), gyroscope, light and proximity using this approach

Example Step Counter App

- **Goal:** Track user’s steps, display it in TextView
- **Note:** Phone hardware must support step counting



```java
package com.starboardland.pedometer;

import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class CounterActivity extends Activity implements SensorEventListener {

    private SensorManager sensorManager;
    private TextView count;
    boolean activityRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        count = (TextView) findViewById(R.id.count);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    }
```

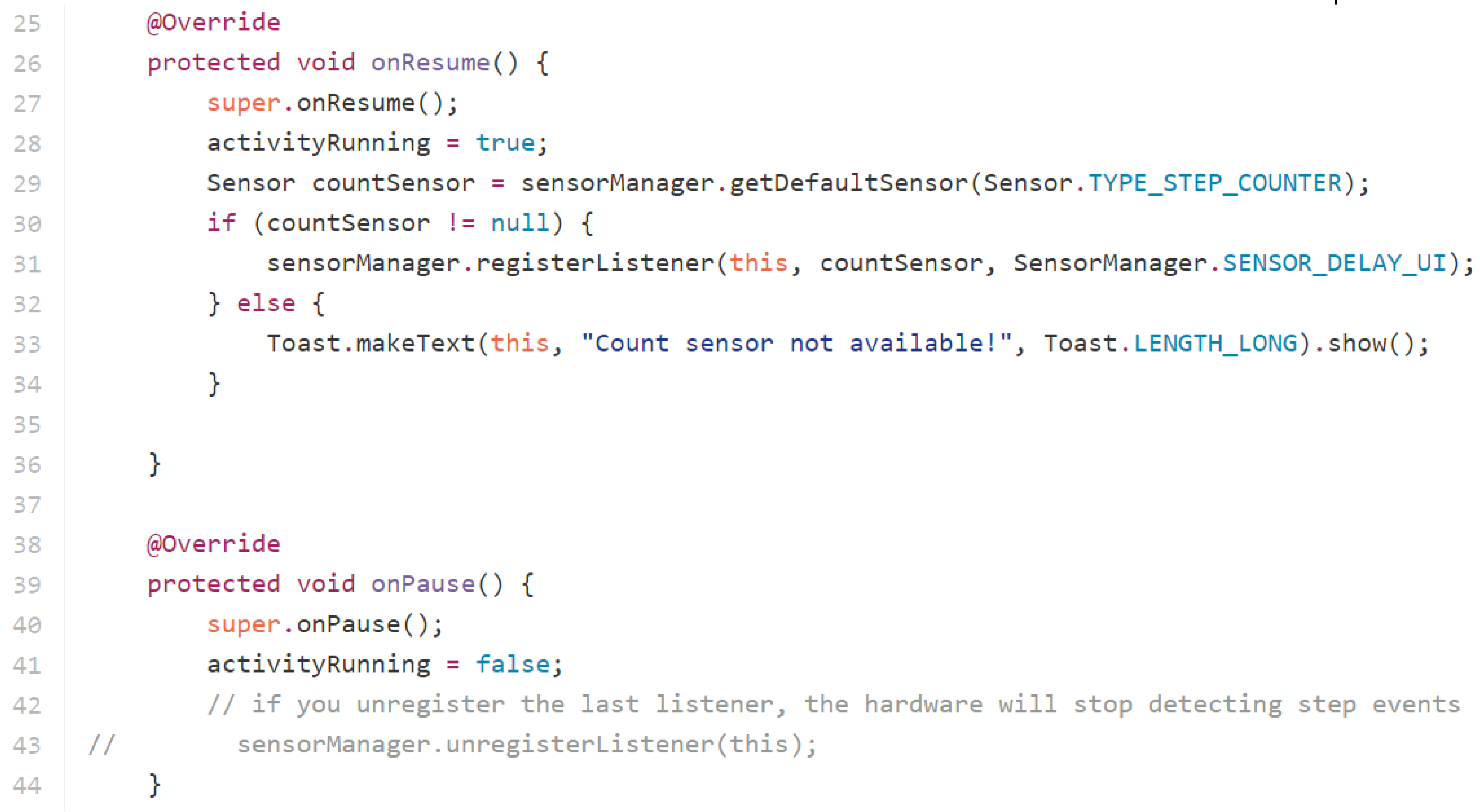


Step count since reboot:

154.0

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/

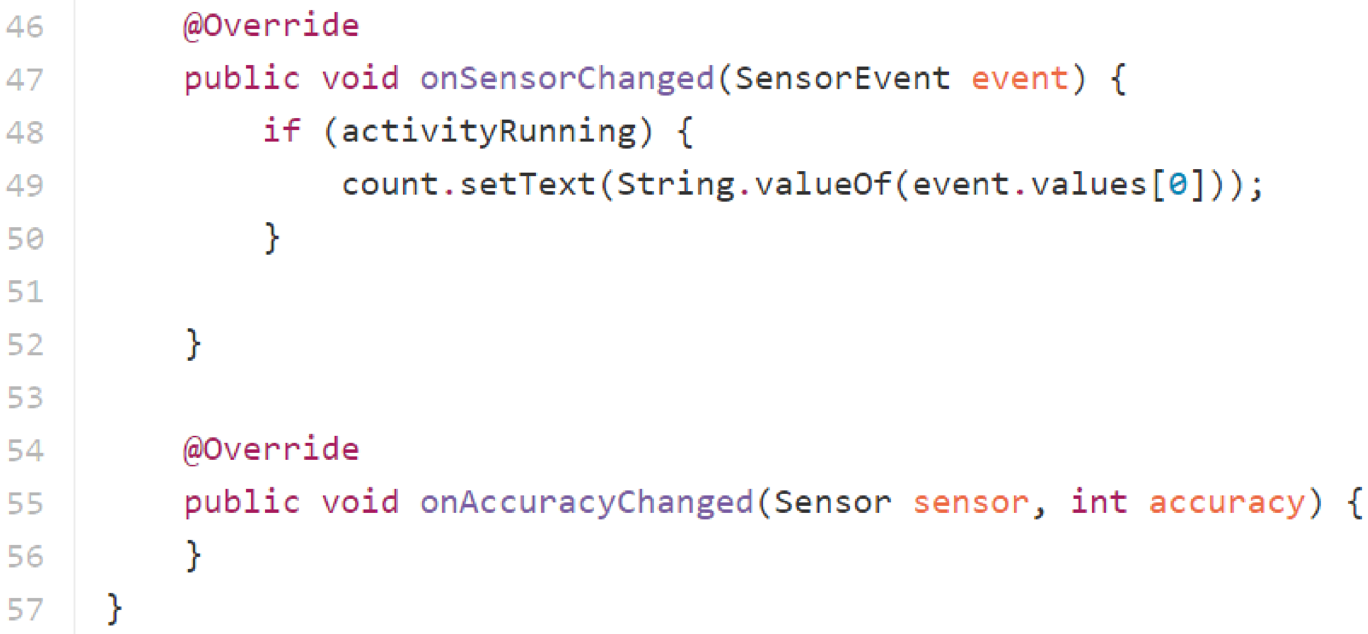
Example Step Counter App (Contd)



```
    @Override
    protected void onResume() {
        super.onResume();
        activityRunning = true;
        Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
        if (countSensor != null) {
            sensorManager.registerListener(this, countSensor, SensorManager.SENSOR_DELAY_UI);
        } else {
            Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_LONG).show();
        }

    }

    @Override
    protected void onPause() {
        super.onPause();
        activityRunning = false;
        // if you unregister the last listener, the hardware will stop detecting step events
//        sensorManager.unregisterListener(this);
    }
```

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/

Example Step Counter App (Contd)



```
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (activityRunning) {
            count.setText(String.valueOf(event.values[0]));
        }

    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
}
```

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/

Best Practices for Sensor Usage

1. **Unregister sensor listeners:** when done using sensor or when app is paused
   - Otherwise sensor continues to acquire data, draining battery

2. **Don't test sensor code on emulator**
   - Must test sensor code on physical device, emulator doesn't support sensors

Best Practices for Sensor Usage (Contd)

3. **Don’t block onSensorChange( ) method:**
   - Android system may call onsensorChanged( ) often
   - So… don’t block it
   - Perform any heavy processing (filtering, reduction of sensor data) outside **onSensorChanged( )** method

4. **Avoid using deprecated methods or sensor types:**
   - E.g. TYPE_TEMPERATURE sensor type deprecated, use TYPE_AMBIENT_TEMPERATURE sensor type instead

Best Practices for Sensor Usage (Contd)

5. **Verify sensors before you use them:**
   - Don’t assume sensor exists on device, check first before trying to acquire data from it

6. **Choose sensor delays carefully:**
   - Sensor data rates can be very high
   - Choose delivery rate that is suitable for your app or use case
   - Choosing a rate that is too high sends extra data, wastes system resources and battery power

Widget Catalog

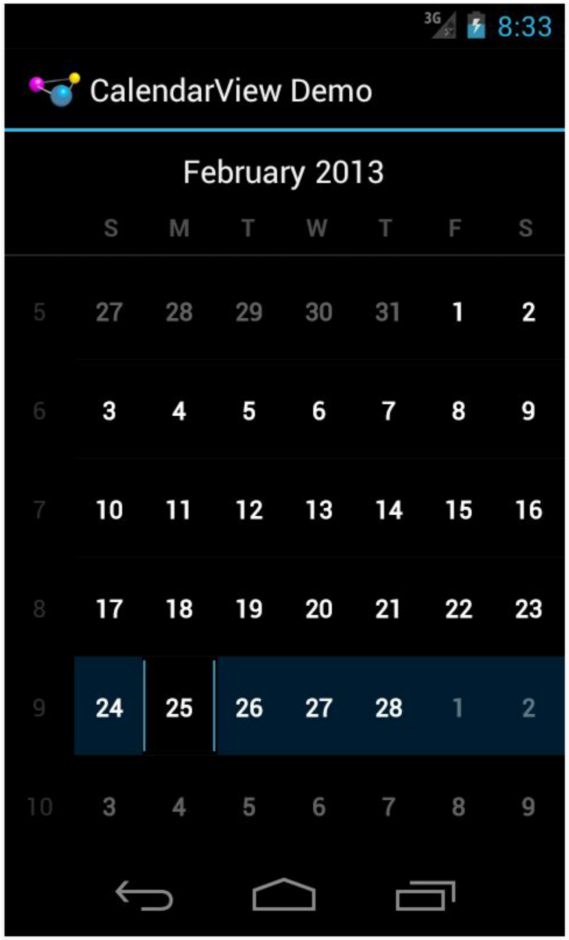
What Widget Catalog?

- Several larger widgets are available
- Can use easily just like smaller widgets, to make your apps look nice and professional
- Several described in Busy Coders book section “**Widget Catalog**”**,** code available
- Examples:
    - CalendarView
    - DatePicker
    - TimePicker
    - SeekBar
- Will not explain coding here. Check book

CalendarView

- Allows user pick a date from a displayed calendar

- Sample project from Busy Coders book:
  - **WidgetCatalog/CalendarView**



8:33

CalendarView Demo

February 2013

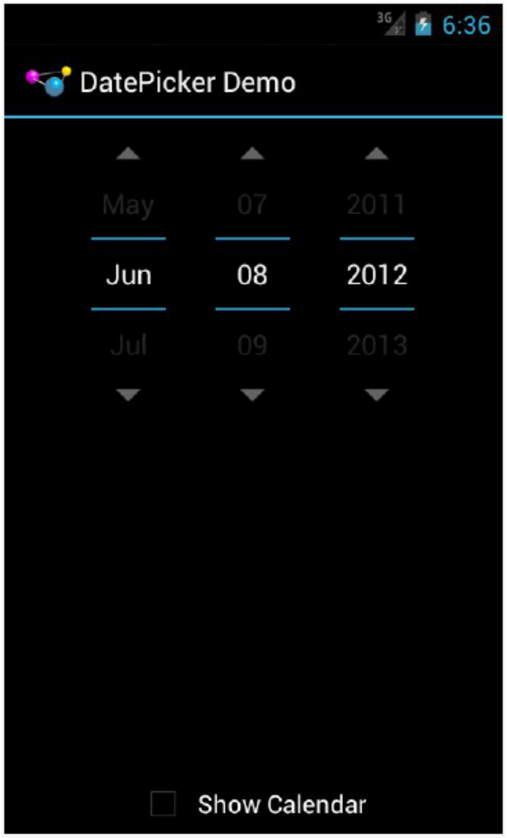
	S	M	T	W	T	F	S
5	27	28	29	30	31	1	2
6	3	4	5	6	7	8	9
7	10	11	12	13	14	15	16
8	17	18	19	20	21	22	23
9	24	25	26	27	28	1	2
10	3	4	5	6	7	8	9

CalendarView Android 4.0

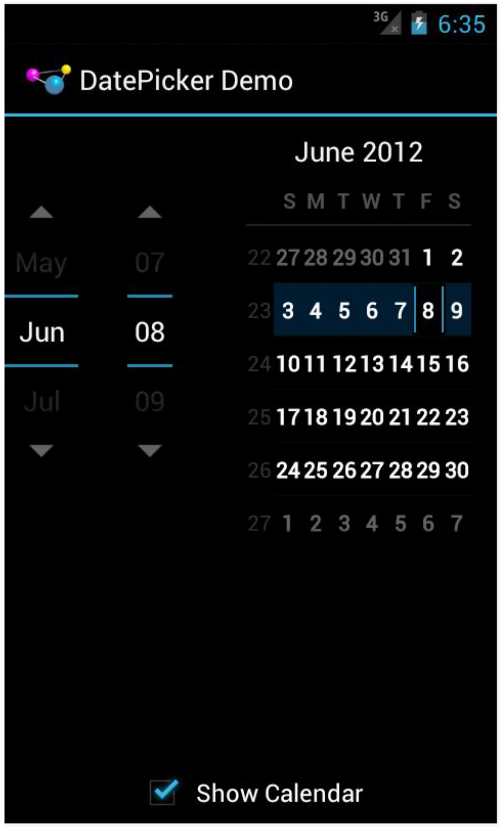
DatePicker

- Allows user pick a date
- Uses date wheel
- Can optionally display a CalenderView as well

- Sample project from Busy Coders book:
  - WidgetCatalog/DatePicker

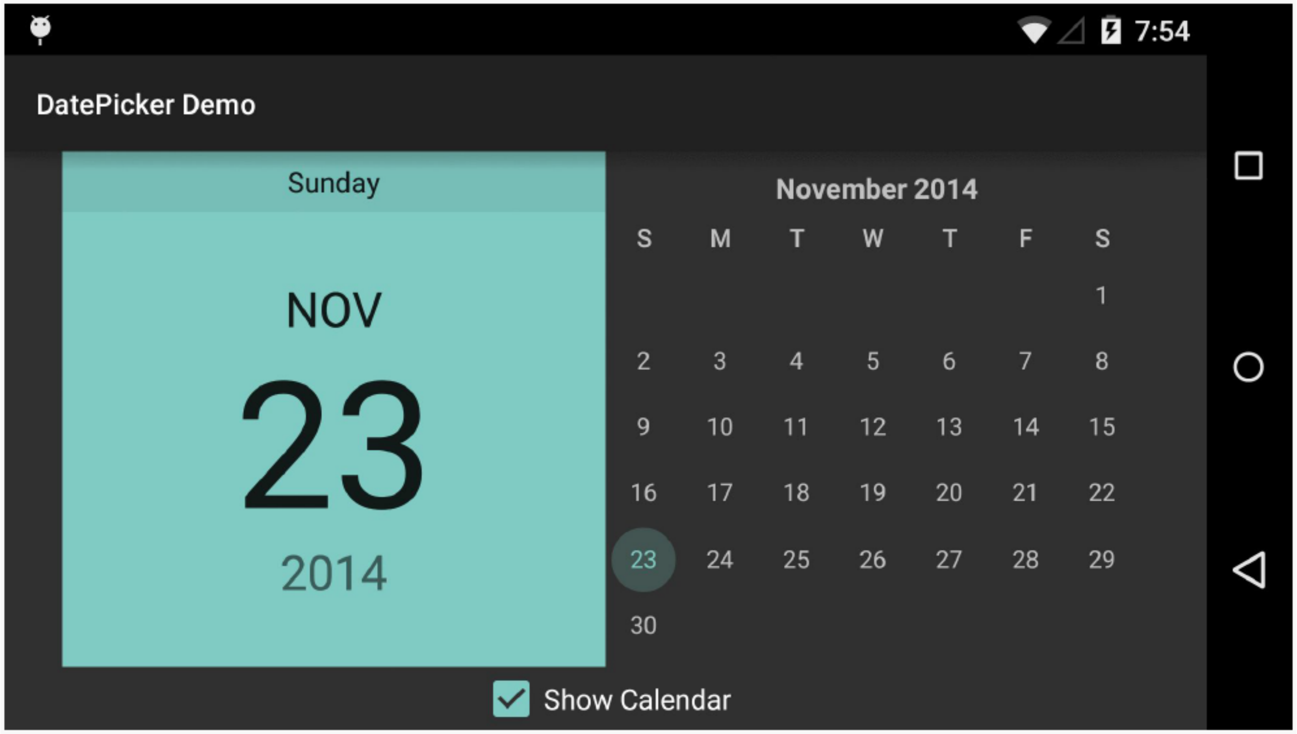


DatePicker without CalendarView Android 4.0



DatePicker with CalendarView Android 4.0

DatePicker



7:54

DatePicker Demo

Sunday

NOV

23

2014

November 2014

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

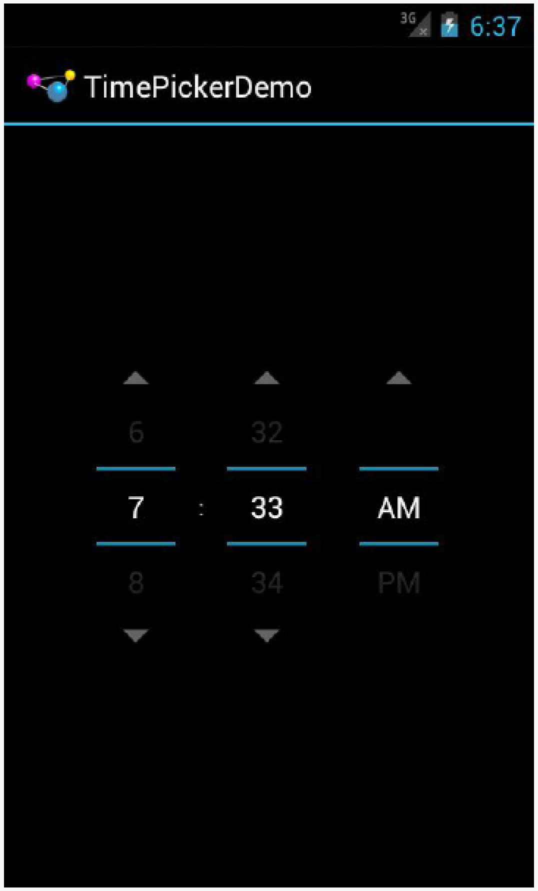
☑ Show Calendar

DatePicker with CalendarView Android 5.0, landscape

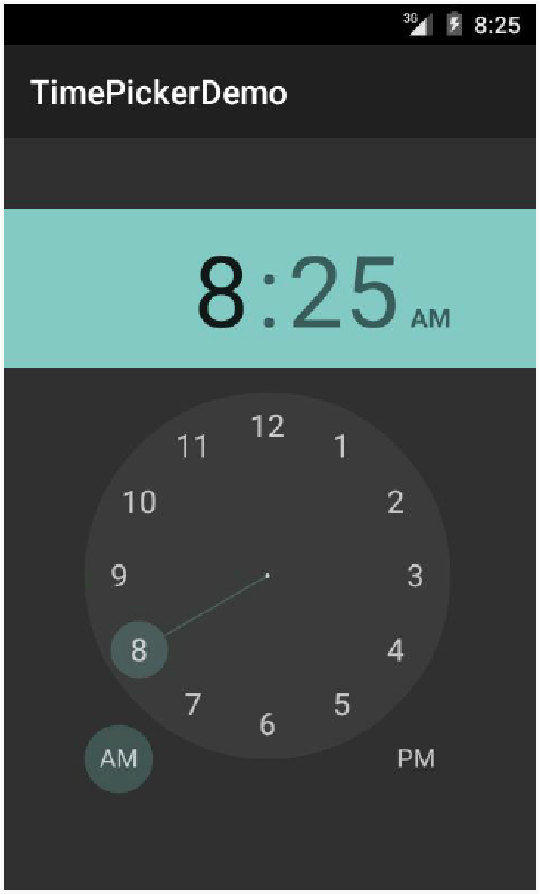
TimePicker

- Allows user pick a time

- Sample project from Busy Coders book:
  - **WidgetCatalog/TimePicker**



TimePicker Android 4.1



TimePicker Android 5.0

Class Timeline: What’s left?

What’s left?

- Total of 15 papers on class website assigned
  - Each group will present 1 paper (tag team style)
- 4 classes during which students will present papers
  - 3 or 4 student talks per class
- Timeline:
  - **Apr 6:** I will talk
  - **April 9, 13:** Student Talks
  - **Apr 16:** Final project proposal (all 15 groups)
  - **Apr 27, 30:** More student talks
  - **May 4:** Final project presentation + submissions

Presentation Guidelines

Overview

- No class Mon Apr 20 (patriots day) and Apr 23 (proj)
- Starting next Thursday, new phase of class
  - Paper presentations
  - Writing critiques of papers
  - Final project: brainstorming, proposal, implement, submit
- Each class, 3 or 4 groups present
- All **students not presenting each class**, pick ANY 1 of the papers presented that class and write critiques of them
- Critiques MUST be submitted (via turnin) BEFORE paper presented
- Next, I provide guidelines on presenting papers and writing critiques

Your Presentation

- About 20 mins
- Estimate: about 2 mins per slide
- About 12 slides should be enough **excluding front page and references**
- Allow 10 mins for questions, discussions => participation points
- **Prepare slides using powerpoint template on course site**

Main Points Presentation Should Cover

- Introduction/motivation:
  - What was main problem addressed by the paper?
  - Why is the problem solved important
- Vision:
  - How will the solution be used eventually?
  - How will this new approach save time, resources, incovenience, etc?

Main Points Presentation Should Cover

- Related Work:
  - What else has been done to solve this problem?
  - How is the approach proposed in this paper different or novel?
    - New approach:
    - New algorithm
    - New technique
    - New experiments
  - Focus on:
    - scientific results, what was learned
    - Engineering results: new design + justification for choices

Main Points Presentation Should Cover

- Methodology/Approach:
  - Summarize the approach/design
  - Describe the implementation used
  - State any assumptions of the authors and limitations of the proposed work
  - What are the design tradeoffs?

Main Points Presentation Should Cover

- Results:
  - Present a **few of the most significant** results/graphs
  - Results/graphs should show how well proposed approach worked or findings
  - Do the presented results back up the claims of the authors?

Main Points Presentation Should Cover

- Discussions/Conclusions/Future Work
  - Summarize what was achieved
  - What did you learn from this paper?
  - What extensions do the authors plan for future work?

Critique Guidelines

Critique Guidelines

- Capture key points of paper
- Half a page max, submitted through turnin
- Don’t just cut-and-paste abstract blindly
- In a year’s time, summary should recall key aspects of paper, refresh memory without re-reading paper
- Provide key important details:
  - Problem, idea, concepts, algorithms tools proposed?
- See guidelines on course website

Critique Guidelines (Contd)

- Assumptions made.
- Design trade-offs?
- How is the organization of the paper, clarity of writing?
- Did the graphs, results support the claims by authors?
- What was good? Bad about paper?
- Suggestions for improvement?
- **Similar app that you know of**

References

- Busy Coder’s guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014